General Disclaimer

One or more of the Following Statements may affect this Document

- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.

- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.

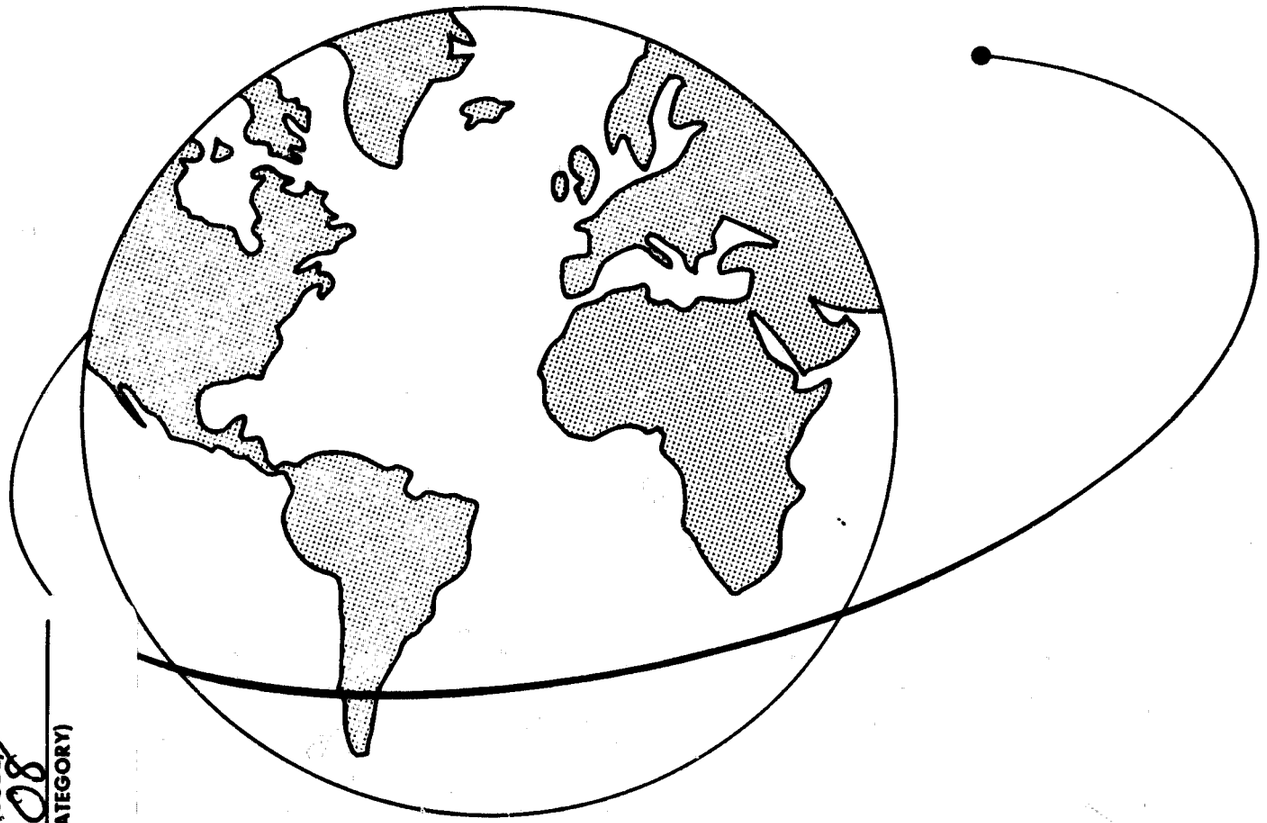
- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.

- This document is paginated as submitted by the original source.

- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

SMITHSONIAN PACKAGE FOR ALGEBRA AND SYMBOLIC MATHEMATICS

N. M. HALL and J. R. CHERNIACK



Smithsonian Astrophysical Observatory
SPECIAL REPORT 291

Research in Space Science
SAO Special Report No. 291

SMITHSONIAN PACKAGE
FOR
ALGEBRA AND SYMBOLIC MATHEMATICS

Norton M. Hall and Jerome R. Cherniack

January 29, 1969

Smithsonian Institution
Astrophysical Observatory
Cambridge, Massachusetts 02138

905-112

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

ABSTRACT

SPASM is a system for computer processing of algebraic expressions. This report assumes familiarity with the FORTRAN language only. Starting from that point, it attempts to explain the function of SPASM by analogy with FORTRAN and to justify the system as an extension of FORTRAN. The text provides only that information necessary to use SPASM as given. Other documents explain how SPASM works and provide maintenance documentation.

RÉSUMÉ

Le SPASM est un système pour traitement par ordinateur des expressions algébriques. Ce rapport suppose que le lecteur est familier avec le langage FORTRAN seulement. Sur cette base, il essaye d'expliquer le rôle du SPASM par analogie avec le FORTRAN et de justifier le système en tant qu'extension du FORTRAN. Le texte fournit seulement l'information nécessaire pour employer le SPASM tel quel. D'autres documents expliquent comment fonctionne le SPASM et fournissent la documentation permettant d'assurer une bonne marche.

КОНСПЕКТ

СПАЗМ'а является системой для обработки алгебраических выражений с помощью электронной вычислительной машины. Этот справочник предполагает хорошую осведомленность только с языком ФОРТРАН. Исходя из этой точки, он пытается объяснить функцию СПАЗМ'ы путем аналогии с ФОРТРАН'ом и подтвердить что эта система является развитием ФОРТРАН'а. В тексте приводятся только те сведения которые необходимы для пользования СПАЗМ'ой. Прочая литература объясняет работу СПАЗМ'ы и предоставляет необходимые руководства для ухода.

SMITHSONIAN PACKAGE FOR ALGEBRA AND SYMBOLIC MATHEMATICS

Norton M. Hall and Jerome R. Cherniack

1. INTRODUCTION

1.1 Higher Level Computer Language

SPASM is one of several efforts in the field of using computers to process other than strictly numerical information. This field is referred to loosely as "symbol manipulation" when it is intended to encompass a broader field than mathematics, and "algebraic manipulation" when it is restricted to familiar forms of mathematical expressions. SPASM is an example of the latter; other well-known efforts in the same area are the IBM system FORMAC, and the Bell Labs system ALPAK.

SPASM is designed as a set of FORTRAN subroutines and functions and for that reason assumes familiarity with FORTRAN. FORTRAN may also serve as the most familiar example of a higher level programming language. FORTRAN may be considered from different points of view: e. g., as a compiler and as an artificial language. The primary concern of a compiler is to eliminate repetitious and trivial coding problems for the programmer. However, to accomplish this end a language must be designed, and into that design other concerns enter. Problem solving is also optimized by making the computer language correspond as closely as possible to the natural language of the user. For the area under consideration, this is conventional mathematical notation.

This work was supported in part by grant NGR 09-015-002 from the National Aeronautics and Space Administration.

Parts of FORTRAN, notably the input-output statements, deal with problems peculiar to computers and unrelated to mathematics. The heart of FORTRAN, as an example of communication between a mathematician and a computer, is the arithmetic replacement statement. The arithmetic replacement statement is designed to look like an equation with an algebraic expression on the right side but only a single variable on the left side. The function of the statement is to evaluate the expression on the right and to set the variable on the left equal to the value. From one point of view, FORTRAN may be taken as an example of an algebraic manipulator. It is possible to represent an algebraic expression that the computer will do something useful with, namely, evaluate it for any values of the variables. However, only the FORTRAN compiler deals with expressions. The compiled program itself typically processes only numbers. If we make the conventional distinction between program and data, the FORTRAN deck is data in the form of algebraic expressions to the compiler. In the user's program, however, the FORTRAN deck represents the program, and without a great deal of work, the only data that program will accept are numbers. FORTRAN has proved immensely valuable to mathematicians, but it may be called primitive in the context of algebraic manipulation because the only operation it can perform is evaluation. Although user-written programs have dealt with problems from all branches of mathematics, they are obliged to find numerical solutions. The only techniques available to the FORTRAN programmer are those of simple arithmetic.

The following sample programs illustrate these ideas. In the sample SPASM programs, the actual calling sequences are glossed over until they can be defined in an orderly manner.

The first program illustrates how the FORTRAN compiler could be used as an algebraic manipulator whose only function is to evaluate expressions. Assume that the values assigned to the variables are always the same and that what we wish to change from job to job is the expression to be evaluated:

```
      X = 3.
      Y = 2.
C     INSERT Z AS A FUNCTION OF X AND Y.
      Z = X**2 + Y**2
      PRINT 1, Z
1     FORMAT (*0Z = *, E20.13)
      STOP
      END
(end-of-file card)
```

In this program, the data (namely, the expression to be evaluated) are punched into a card and inserted into the deck after the comment card.

A SPASM program to do the same thing might look like this:

```
      L = LREAD (...)
          .
          .
          .
      set up tables for EVAL
          .
          .
          .
      Z = EVAL (L,...)
      PRINT 1, Z
1     FORMAT (*0Z = *, E20.13)
      STOP
      END
      (end-of-record card)
      Z = X**2 + Y**2 $
      (end-of-file card)
```

In both programs, the data to be processed are an expression that can have different values from run to run. The FORTRAN program is designed in a peculiar way, because it is the compiler, and not the user's program, that has the capability to operate on algebraic expressions.

The following example illustrates the conventional use of FORTRAN. Here, the program (that is, the expressions to be evaluated) is held constant, and the values of the variables are changed from run to run.

```
      READ 1, X
      READ 1, Y
1     FORMAT (F10.0)
      Z = X**2 + Y**2
      PRINT 2, Z
2     FORMAT (*0Z = *, E20.13)
      STOP
      END
      (end-of-record card)
      3.
      2.
      (end-of-file card)
```

A SPASM program to do, not the same thing, but something analogous might be as follows:

```
      L = LREAD (...)
      L2 = LREAD (...)
      L3 = LSUM (L, L2)
      CALL LPRINT (L3,...)
      STOP
      END
      (end-of-record card)
      X = 2A + 3B $
      Y = 3A + 4B $
      (end-of-file card)
```

The output of the program might look like

```
Z = 5A + 7B    .
```

These examples illustrate that at the level of the user's program, FORTRAN programs deal with numerical data, whereas SPASM programs deal with data in the form of algebraic expressions. SPASM performs on its kind of data operations analogous to those FORTRAN performs on numbers, in the last case adding two items and printing the sum.

FORTRAN may be called sophisticated in the sense that much effort has been brought to bear to make it do what it does elegantly and economically. SPASM also evaluates expressions but does it very uneconomically indeed, compared with FORTRAN. SPASM may be said to be less primitive than FORTRAN only in the sense that it extends the range of operations available to the programmer. A good example is that the programmer can differentiate an expression and then evaluate the derivative. A numerical solution would involve an approximation by difference methods requiring considerable work as well as worry about error.

SPASM, in turn, is primitive compared to what might be done in the way of algebraic manipulation. Briefly, SPASM allows the programmer to produce the sum, difference, product, and quotient of two expressions; to differentiate an expression; to substitute an expression in another; to evaluate expressions; and to transform expressions in various ways with respect to bracketing. There are also special provisions for rational polynomials and Poisson (trigometric) series. SPASM expressions are broader than FORTRAN expressions in that they include symbolic derivatives of the form dy/dx, but less broad in that exponents are restricted to positive and negative integers.

1.2 SPASM's Overall Design

From the user's point of view, SPASM is a set of FORTRAN subroutines and functions. In general, various housekeeping tasks are performed by subroutine calls, and the real work of the system is done by function calls. Most of these functions have mnemonic names preceded by the letter L, to force them all to "type" integer. They take expressions and variables as their arguments and return a new expression as their value. This convention

was adopted for two reasons. First, it allows several operations to be nested in a single statement. Second, it emphasizes to the programmer that the arguments of an operation are left unchanged and that a new expression is created. Some SPASM functions do not return expressions as output, e.g., EVAL, which returns a floating-point number.

SPASM is based on SLIP, a list-processing system. List processing is a technique in which the elements of a set are ordered by a system of pointers rather than by physical location. The advantage of this technique, which is the reason for its adoption here, is that the allocation of storage is fully dynamic. Any word of storage can be attached to any set of elements, and the storage used by sets of elements can be returned to the pool of available words. In SPASM, an expression is stored as a list, and the user is provided with a handle called its name. The name of an expression is a FORTRAN variable containing a pointer to the origin of the list containing the expression. When the user no longer needs an expression and wishes to return it to the pool of free words, he codes

```
CALL ERASE (L)
```

where L is the name of the expression to be erased. In general, it is necessary to plan the erasure of intermediate results with care, to avoid running out of storage.

The variables that appear in SPASM expressions are referred to as atomic variables. The internal representation of an atomic variable is essentially that of a Hollerith constant put into a standard format, with some housekeeping information. The user obtains the proper format by calling the function ATOM with the Hollerith representation of the variable. For example,

```
XVAR = ATOM (1HX)
```

stores in XVAR the SPASM representation of the variable X. Whenever a SPASM function requires a variable as an argument, this standard form

must be provided. For example, LDERIV, which produces the derivative of an expression, takes as arguments the name of the expression to be differentiated and the variable of differentiation.

```
L2 = LDERIV (L, ATOM (1HX))
```

will cause the expression whose name is in L to be differentiated with respect to X, and the name of the new expression to be stored in L2.

SPASM expressions are analogous to the data of a conventional FORTRAN program. They originate on data cards separate from the program, from which they are read by the function LREAD. Facilities are provided to write them on tape and read them back and to print them on the output file. There is also a facility to punch them into FORTRAN cards so that they can be compiled as arithmetic replacement statements. An analog of constants in a FORTRAN program is provided by the function LDECODE. SPASM expressions on data cards look like FORTRAN arithmetic replacement statements. However, the left-hand side is nonessential. The right-hand side is what SPASM works on. The principal qualifications of this rule-of-thumb are that SPASM expressions must be terminated by $, may not contain subscripted variables, and may contain only integer exponents. The appendix contains a more systematic statement of the syntax.

Before any SPASM routine is used, it is necessary to CALL INIT. This subroutine initializes SLIP and various bookkeeping arrangements. Practically every SPASM program must then CALL RDEF. This subroutine reads control cards containing definitions. If functions appear in the SPASM expressions, they must be defined on these cards, and if they are to be differentiated, the derivatives of the functions with respect to their arguments must be supplied. SPASM does not assume the existence of any function, not even SIN and COS. In addition, variables may be defined as depending implicitly on other variables. The formats of these control cards are given in a later section. A typical SPASM program continues by reading expressions from cards and producing other expressions. Calls to the SPASM functions will

be interspersed with calls to LPRINT, which prints expressions and calls to ERASE. It is advisable to print intermediate results liberally and to erase intermediate results early. At the end of a program, one should erase all remaining expressions and then call MOPUP and LISTS to see if everything has, in fact, been erased. This is not essential for a simple program but is most important in debugging a subroutine. Subsequent sections of this manual define the calling sequences and effects of each of the SPASM routines. Many of the details are deferred to a later section on housekeeping. Appendices contain notes on loading the system under 6400 SCOPE and a complete sample program from JOB card to end-of-file.

2. INPUT-OUTPUT OPERATIONS

The input and output routines are defined together because they interact to a considerable degree. There are two formats for expressions, which might be called the input and print formats. The difference is in the treatment of exponents. In the input format, exponentiation is indicated by **, as in FORTRAN. In the print format, exponents are printed on the line above the main line. However, it is also possible to output expressions in the input format in order to allow expressions to be stored on tape.

It is necessary here to distinguish between the label of an expression and its name as previously defined. The input-output routines provide for reading and writing labels that appear on the left side of an =. The user may wish to make his names and labels agree, but there is no necessary connection between them. Labels are merely Hollerith fields that the user may find helpful in keeping track of his expressions, but the system does not depend on them in any way.

Following are several examples of well-formed expressions on data cards:

```
GAMMA = X**2 + Y**2  $                                    (1)

PSI = A*SIN (B) - B/COS (A)  $                            (2)

EXP3 = (ALPHA + BETA)*(1 - GAMMA**2)
       /(3 + DELTA)  $                                    (3)

1/2 + 2/3*X - 3/4*X**2 + 4/5*X**3  $                      (4)
```

In these examples, GAMMA, PSI, and EXP3 are recognized as labels because they are followed by =. They are returned to the user as left-justified Hollerith constants, equivalent to 5HGAMMA, 3HPSI, and 4HEXP3. The label of expression (4) is returned as all blanks. Expression (3) illustrates that expressions continue from card to card until a $ is found.

```
NAME = LREAD (LABEL, OK)
```

is the calling sequence for reading one expression from data cards. If the data cards listed above were positioned next in the input file and this statement were executed, then after execution NAME would contain the name of the expression $X^2 + Y^2$, LABEL would contain 5HGAMMA, and OK would have the logical value TRUE, meaning that no error was discovered. If an error were discovered, for example, if parentheses were unbalanced, then OK would be given the value FALSE. However, NAME would contain the name of as much of the expression as was well formed, and the input file would be positioned properly to read the next expression. Thus, it is possible to diagnose all data errors before a job is terminated.

```
NAME = LRDTAPE (LABEL, OK, LUN)
```

works the same way, but reads from the file specified by LUN, presumably a tape or disk file.

```
NAME = LDECODE (LABEL, OK, 4H1+X$)
```

reads an expression from a Hollerith constant in the FORTRAN code. In this case, NAME gets the name of the expression 1 + X, LABEL gets blank, and OK is set TRUE. LDECODE is restricted to a Hollerith field of no more than 100 characters.

```
CALL LPRINT (NAME, IFIGS, NLABEL, LABEL)
```

is the calling sequence to cause an expression to be printed on the output file in print format. NAME contains the name of the expression to be printed. IFIGS contains the number of significant figures to be printed for floating-point numbers. NLABEL contains the number of characters in LABEL. LABEL contains a label to be printed before an =. If NLABEL contains 0,

LABEL may be omitted. Suppose the input cards listed above were positioned next in the input file, and the following code were executed:

```
      DO 10 I = 1, 4
      NAME (I) = LREAD (LABEL (I), OK)
      IF (OK) GO TO 10
      PRINT 1
1     FORMAT (*0ERROR.*)
10    CALL LPRINT (NAME (I), 6, 5, LABEL (I))
      STOP
```

The output file would look like this:

$$\text{GAMMA} = X^2 + Y^2$$

$$\text{PSI} = A*\text{SIN}(B) - B/\text{COS}(A)$$

$$\text{EXP3} = (\text{ALPHA} + \text{BETA})*(1 - \text{GAMMA}^2)/(3 + \text{DELTA})$$

$$= 1/2 + 2/3*X - 3/4*X^2 + 4/5*X^3 .$$

LPRINT works with lines equal to three print lines: a blank line, an exponent line, and a main line. It leaves a margin on the left equal to NLABEL + 3 columns and prints LABEL in the margin of the first line. If NLABEL = 0, no margin is left, and = does not appear.

CALL LWRITE (NAME, IFIGS, NLABEL, LABEL, LUN)

causes the expression to be written on file LUN in the input format, virtually a copy of the input cards.

CALL LFORT (NAME, IFIGS, NLABEL, LABEL, LUN)

causes the expression to be written on LUN in a format compatible with FORTRAN. The intention is to produce cards that can be inserted in a FORTRAN deck to evaluate the expressions. Notice that the left side of these arithmetic replacement statements is supplied by LABEL.

CALL LENCODE (NAME, IFIGS, NLABEL, LABEL, LUN, BUF, I, J, KT)

acts like LPRINT except that instead of being printed, the expression is encoded into BUF, a two-dimensional array of I rows and J columns. The number of columns used is returned in KT. KT is accurate even when it exceeds J, but then the last KT-J columns are not transmitted. LENCODE sets all of BUF to blanks before executing. LUN is ignored, as in LPRINT. LENCODE uses three columns of BUF for each line of expression, just as LPRINT uses three print lines: a line of blanks, a line of exponents, and a main line.

For the remainder of the operations, the following convention will be adopted. The circumlocution "the expression whose name is contained in" will be dropped, and the names of expressions will be used as though they were the expressions themselves. Each operation will be introduced by its calling sequence, and its effect will be described briefly and illustrated with sample expressions. The value of an expression will be given as follows:

$$\text{NAME} = X^2 + Y^2 \quad .$$

Notice that this is not intended to correspond to a data card, a print line, or a FORTRAN statement. The reader should clearly distinguish at this point between names and labels and between SPASM expressions and FORTRAN expressions.

3. OPERATIONS ON POLYNOMIALS

3.1 Basic Operations

Certain rules about the way expressions are handled are of particular importance for the arithmetic operations. One is that numeric coefficients are maintained as rational fractions reduced to lowest terms whenever possible. The exceptions occur when coefficients are read as floating-point numbers with fractional parts or when either the numerator or the denominator becomes too large to be held in one word. Whenever a rational number combines with a floating-point number, the result is a floating-point number.

The other rule is that similar terms are always collected. Furthermore, in order to achieve this economically, all expressions are rearranged in a standard way. The factors of each term are put in a standard order to make it easier to recognize similar terms. Also, the terms of each expression are put in a standard order so that the terms of a new expression can be collected as they are produced. This design goes far to alleviate the problem of "intermediate swell." This chronic problem in the field of algebraic manipulation refers to the fact that intermediate results may occupy much more storage than does the final result. The user's only concern is to understand why his expressions may be printed in quite different form from what he punched into cards.

L = LSUM (L1, L2)

L is the sum of L1 and L2. If

$$L1 = 2/3*y^2 + 1/4*x*y - 3/8*x^2$$

and

$$L2 = 1 + 1.347*x*y - 1/8*x^2 \quad ,$$

then

$$L = 1 + 2/3*y^2 + 1.597*x*y - 1/2*x^2 \quad .$$

Notice that because one of the coefficients of xy in the operand is a floating-point number, the coefficient of xy in the result is, too. Also notice that the coefficient of x^2 is reduced to lowest terms.

L = LDIFF (L1, L2)

L is the result of subtracting L2 from L1. If

$$L1 = 1/3 + 1/4*x + x*y$$

and

$$L2 = 1/4 + 3y + 2x*y \quad ,$$

then

$$L = 1/12 - 3y + 1/4*x - x*y \quad .$$

If

$$L1 = 2*y/(1 - x)/(1 + x) \quad ,$$

and

$$L2 = y/(1 + x) \quad ,$$

then

$$L = 2*y/(1 - x)/(1 + x) - y/(1 + x) \quad .$$

L = LPROD (L1, L2)

L is the product of L1 and L2. If

$$L1 = 1/2*z + 1/3*y + 1/4*x$$

and

$$L2 = 1/2*z - 1/3*y \quad ,$$

then

$$L = 1/4*z^2 - 1/9*y^2 + 1/8*x*z - 1/12*x*y \quad .$$

L = LQUOT (L1, L2, EXACT)

L is the result of dividing L1 by L2. EXACT is returned with the value TRUE if the division could be performed exactly, i. e., if L2 is a factor of L1. If not, EXACT is returned FALSE, and L is formed by distributing the inverse of L2 over the terms of L1. If

$$L1 = x^2 - y^2$$

and

$$L2 = x + y \quad ,$$

then

$$L = x - y, \text{ EXACT} = .T. \quad .$$

If

$$L1 = x^2 + y^2$$

and

$$L2 = x + y \quad ,$$

then

$$L = x^2/(x + y) + y^2/(x + y), \text{ EXACT} = .F. \quad .$$

L = LGCD (L1, L2)

L is the greatest common divisor of L1 and L2. The exponents in L1 and L2 not enclosed in brackets must be positive integers. If

$$L1 = (x + 1/y)^2 - A^2$$

and

$$L2 = (x + 1/y)^2 - 2*A*(x + 1/y) + A^2 \quad ,$$

then

$$L = (x + 1/y) - A \quad .$$

L = LDERIV (L1, VAR)

L is the result of differentiating L1 with respect to VAR, an atomic variable produced by ATOM, as in VAR = ATOM (1HX). If

$$L1 = y^2 + 2*x*y + x^2$$

and

$$VAR = ATOM\ (1HX) \quad ,$$

then

$$L = 2*y + 2*x \quad .$$

If L1 contains functions, the user must have supplied models of the derivatives of the functions with respect to each of their arguments. The control cards to accomplish this are defined in the section on initialization. These models are SPASM expressions in which the first argument is represented by the atomic variable ARGA, the second by ARGB, and so on. For example, the model for the derivative of SIN with respect to its argument would be the expression COS (ARGA). The derivative of a function is formed by substituting its actual arguments in these models and multiplying by the derivatives of the arguments (see Section 6.2). In general,

$$dF(y_1, \ldots, y_n)/dx = \partial F/\partial y_1 \; dy_1/dx + \ldots + \partial F/\partial y_n \; dy_n/dx \quad .$$

If

$$L1 = SIN\ (x^2)$$

and

$$dSIN/dARGA = COS\ (ARGA) \quad ,$$

then

$$L = 2*x*COS\ (x^2) \quad .$$

The user may define atomic variables as depending implicitly on one or more other atomic variables. The derivatives of dependent variables are represented in the form

$$DERIV\ (y, x, 1, z, 2) = \frac{\partial^3 y}{\partial x \partial z^2} \quad .$$

This form differs from a function in that its arguments can be only atomic variables and integers and are variable in number. It is defined by SPASM rather than by the user and is treated specially. If

$$L1 = y + DERIV\ (y, z, 1)$$

and

$$VAR = ATOM\ (1HX) \quad ,$$

then

$$L = DERIV\ (y, x, 1) + DERIV\ (y, z, 1, x, 1) \quad .$$

It is possible to substitute an expression for a dependent variable, causing the indicated differentiation to be performed at that time. An example is given in the definition of LSUBST.

L = LINTEG (L1, VAR)

L1 is an expression not containing VAR in brackets or occurrences of VAR^{-1}. (Caution: This routine produces incorrect results if either condition is violated.) VAR is in ATOM format and

$$L = \int L1\ d(VAR) \quad .$$

If

$$L1 = 2*y + 2*x + 4x^{-2}$$

and

VAR = ATOM (1HX) ,

then

$$L = 2*x*y + x^{2} - 2*x^{-1} \quad .$$

To copy an expression,

L = LSTCPY (L1)

makes L a copy of L1.

3.2 More Complicated Operations: Brackets

In FORTRAN coding, brackets are used to define the range of operators and may be used to control the order in which an expression is evaluated. Redundant brackets have no effect at all. In the context of algebraic manipulation, the significance of brackets is broader, because they also determine the form of the result. Brackets in the operands may cause the result to be more or less complicated, but in most cases the form will be different. One of the design principles of SPASM is to avoid wasting machine time second-guessing the user. This sometimes causes silly results. For example, the expression X-(X) will not be recognized as equal to zero until the user directs the brackets to be cleared.

The user may anticipate the form of the result pretty well by applying the following rule. In the basic arithmetic operations, a bracketed subexpression is treated exactly as though it were an atomic variable. Care is taken that equal subexpressions of the same form are recognized as identical, but again they are not of the same form if they are bracketed differently. For example, suppose the following statement is executed:

L = LPROD (L1, L2)

If

$$L1 = (1 + x)$$

and

$$L2 = (1 + x) \quad ,$$

then

$$L = (1 + x)^2 \quad .$$

If

$$L1 = (1 + x)$$

and

$$L2 = (1 - x) \quad ,$$

then

$$L = (1 + x)*(1 - x) \quad .$$

If

$$L1 = (1 + x)$$

and

$$L2 = ((1 + x)) \quad ,$$

then

$$L = (1 + x)*((1 + x)) \quad .$$

A later section explains that the interpretation of brackets is more complicated when the operands are expressions of type "rational fraction." The operations LEXPAND and LCLEAR are provided to allow the user to remove brackets to the extent possible. For some operations, e.g., LDERIV and LSUBST, it must be confessed that the form of the result with respect to bracketing is somewhat arbitrary, not to say unpredictable. The user is advised to experiment to get a feeling for the ways this problem is resolved.

3.2.1 Expanding brackets

L = LEXPAND (L1)

L is the result of performing the multiplications implied by bracketing in L1. In the case of expressions in which no subexpression is part of the denominator of a fraction, this eliminates subexpressions, except for the arguments of functions. The arguments of functions are, however, expanded in the same sense.

When subexpressions do occur in the denominators of fractions, it is impossible, in general, to remove all brackets. Among the possible half-measures, the one selected is to expand the denominator of a fraction and distribute it over the terms of the numerator. This alternative was selected by considering the usual motive for expansion, which is to allow terms produced by the expansion of a given term in L1 to combine with similar terms produced by the expansion of other terms in L1. For this purpose to be served, it is necessary that brackets be cleared from the numerators of L and that denominators of equal value be put in identical form. If

$$L1 = (y - x)(y + x) \quad ,$$

then

$$L = y^2 - x^2 \quad .$$

If

$$L1 = z*(x*(1 + y))^2 \quad ,$$

then

$$L = x^2*z + 2*x^2*y*z + x^2*y^2*z \quad .$$

If

$$L1 = (1 + x)/(1 + y) \quad ,$$

then

$$L = 1/(1 + y) + x/(1 + y) \quad .$$

L = LBRAC (L1)

The value of LBRAC is L1 enclosed in brackets. For example, if

L1 = X + 2

and

L2 = Y + 1 ,

then the result of the statement

L3 = LPROD (L1, LBRAC (L2))

is the expression X(Y + 1) + 2(Y + 1).

3.2.2 Substitution

L = LSUBST (L1, VAR, L2)

L is the result of substituting the expression L2 for occurrences of the atomic variable VAR in L1. Operations on VAR, such as multiplication, exponentiation, and differentiation, are carried out on L2, but other brackets in L1 are not expanded. If

L1 = y^2*z ,

VAR = ATOM (1HY) ,

and

L2 = 1 + x ,

then

L = z + 2*x*z + x^2*z .

If

$$L1 = y^2*(1 + z) \quad ,$$

then

$$L = (1 + z) + 2*x*(1 + z) + x^2*(1 + z) \quad .$$

If

$$L1 = \text{DERIV } (y, x, 1, z, 1) \quad ,$$

$$\text{VAR} = \text{ATOM } (1\text{HY}) \quad ,$$

and

$$L2 = z^3 + 3*x*z^2 + 3*x^2*z + x^3 \quad ,$$

then

$$L = 6z + 6x \quad .$$

3.2.3 Evaluation

X = EVAL (L1, VARS, FUNCS, OK)

VARS is a table of atomic variables followed by numeric values to be assigned to each. FUNCS is a table of function names followed by the addresses of entry points of routines that evaluate the functions. If all variables and functions encountered in L1 are included in these tables, then OK is returned true and X contains the value of L1 for these values of the variables. If a variable or function is not found in the table, OK is returned false and X contains indefinite (0/0).

The form of the tables is a linear array in which the odd elements are variable or function names and the following even element is the associated value or entry point. Tables are terminated by a zero word in an odd element. Variable names must be formatted by statements of the form VARS (I) = ATOM (1HX); function names, by statements of the form FUNCS (I) = FNAME (3HSIN). The addresses of entry points can be obtained by statements of the form

```
REAL LOCF
EXTERNAL SIN
.
.
.
FUNCS (I + 1) = LOCF (SIN)
```

The user must supply routines in the form of FORTRAN functions of one argument to evaluate his functions. Where a function has more than one argument, the routine that evaluates it will be called with a linear array in which the i^{th} element is the value of the i^{th} argument. In any case, the user's functions are supplied with floating-point numbers that are the values of the arguments for the values of the variables supplied in VARS.

4. OPERATIONS ON NONPOLYNOMIALS

4.1 Introduction

By using the SPASM functions already described, a user can write routines for combining expressions that are not polynomials. As an illustration, we now write addition and multiplication routines for complex polynomials. We represent a complex polynomial P = Real + i Imag, where Real and Imag are SPASM expressions, by the one-dimensional array DIMENSION P(2) where P(1) = REAL and P(2) = IMAG (ignore type conversion). Then for L3 = L1 + L2, L4 = L1*L2, we have the simple routines:

```
SUBROUTINE CPROD (L1, L2, L4)
DIMENSION L1(2), L2(2), L4(2)
J = LPROD (L1(1), L2(1))
K = LPROD (L1(2), L2(2))
L4(1) = LDIFF (J, K)
CALL ERASE (J)
CALL ERASE (K)
L = LPROD (L1(1), L2(2))
M = LPROD (L1(2), L2(1))
L4(2) = LSUM (L, M)
CALL ERASE (L)
CALL ERASE (M)
RETURN
END
```

```
SUBROUTINE CSUM (L1, L2, L3)
DIMENSION L1(2), L2(2), L3(2)
L3(1) = LSUM (L1(1), L2(1))
L3(2) = LSUM (L1(2), L2(2))
RETURN
END
```

We have not included these particular applications in SPASM because they are not often needed in celestial mechanics and because the user can provide them easily if required.

We have, however, provided special routines for the manipulation of the more difficult cases of rational polynomial fractions and Poisson (trigonometric) series.

Suppose $L_1 = a^{-1} \sin(x)$, $L_2 = b^{-1} \sin(x)$. Then,

$$L_1 + L_2 = a^{-1} \sin(x) + b^{-1} \sin(x) \tag{5}$$

$$= a^{-1}b^{-1} (a \sin(x) + b \sin(x)) \tag{6}$$

$$= (a^{-1} + b^{-1}) \sin(x) \quad , \tag{7}$$

$$L_1 L_2 = a^{-1}b^{-1} \sin^2(x) \tag{8}$$

$$= a^{-1}b^{-1} (1 + \cos 2x)/2 \quad . \tag{9}$$

Lines (5) and (8) correspond to the results that SPASM, as previously described, would produce. It is sometimes useful to collect terms over the least common multiple of their denominators as in line (6); rational fractions are introduced to facilitate such calculations. Manipulation of trigonometric series is often done as in lines (7) and (9), and the trigonometric-series routines perform in this fashion.

Thus, the purpose of having different routines to perform the same function (say, addition) is to easily allow the user different forms of mathematically equivalent expressions. Trigonometric series and rational fractions have been singled out because of their closure properties and their relevance to astronomical calculations.

4.2 Rational Arithmetic

Suppose we wanted to compute the difference between $2Y/(1 - X)(1 + X)$ and $Y/1 + X$. With pencil and paper, we might proceed:

Example 1

$$\frac{2Y}{(1 - X)(1 + X)} - \frac{Y}{1 + X} = \frac{2Y}{(1 + X)(1 - X)} - \frac{Y(1 - X)}{(1 + X)(1 - X)} \qquad (10)$$

$$= \frac{2Y - Y(1 - X)}{(1 + X)(1 - X)} \qquad (11)$$

$$= \frac{Y + XY}{(1 + X)(1 - X)} \qquad (12)$$

$$= \frac{Y}{1 - X} \quad . \qquad (13)$$

Note that this is <u>not</u> what LDIFF (as previously described) does. For if

L1 = 2Y/[(1 - X)(1 + X)], L2 = Y/(1 + X), and L = LDIFF (L1, L2),

then

L = 2Y/[(1 - X)(1 + X)] - Y(1 + X) .

Arithmetic, as performed in Example 1, is often required, and SPASM provides facilities for such computations.

There is one complication: The reduction to lowest terms of the last step of the example involved finding the greatest common divisor of (X + 1)(X - 1) and XY + Y. Finding the greatest common divisor of two arbitrary expressions is surprisingly time consuming. We do two things to minimize this problem.

First, we organize addition and subtraction of rational fractions, so that the reduction to lowest terms is not done. We do only steps (10) to (12) of Example 1. There is a separate function for reduction to lowest terms.

Second, we preserve all we can of the factorization of numerator and denominator. The user can remove parentheses at any time. There follow examples, in ordinary notation, that show how SPASM can operate on rational fractions.

Example 2

$$\frac{2Y}{1 - X^2} - \frac{Y}{1 + X} = \frac{2Y(1 + X) - Y(1 - X^2)}{(1 - X^2)(1 + X)}$$

$$= \frac{Y + 2YX + X^2Y}{(1 - X^2)(1 + X)}$$

Comparison of Examples 1 and 2 shows how it helps to know that $X^2 - 1 = (X - 1)(X + 1)$. Factorization of arbitrary expressions is harder than finding greatest common divisors, so SPASM uses whatever factorization it has.

Example 3

$$\frac{(X + 1)(W + 1)}{(Y + 1)Z^4} \cdot \frac{(Y + 1)^2Z^4}{(W + 1)(X + Y)^2} = \frac{(X + 1)(Y + 1)}{(X + Y)^2}$$

Example 4

$$\frac{1 + \text{SIN}(X)}{\text{SIN}(Y)} \cdot \frac{\text{SIN}^2(Y)}{(1 + \text{SIN}(X))^3} = \frac{\text{SIN}(Y)}{(1 + \text{SIN}(X))^2}$$

We now describe how SPASM recognizes rational fractions and manipulates them.

Just as FORTRAN variables can be of different types (real, integer, complex, etc.), SPASM expressions can be of different types. In FORTRAN, the type of a variable can be established lexigraphically (beginning with I, J, K, L, M, N) or by type declaration (REAL X, INTEGER Y). In SPASM, type is established by default or by explicit call to a SPASM function. All expressions previously described are, by default, type "SPASM Series (SS)." After use of the SPASM functions LRATIO (described below), the resulting expression becomes type "Rational Fraction."

In FORTRAN, the type of the variable affects, during compilation, the object code that is produced by the compiler. In SPASM, the type of an expression is used at execution time by some SPASM functions as a signal to perform in a special way.

```
L = LRATIO (L1)
```

L1 is an expression. L1 is changed into the internal form that SPASM uses for rational fractions, and the result is given type rational fraction.

Routines that give special treatment to rational fractions are LSUM, LDIFF, LPROD, LQUOT, and LDERIV.

The functions below perform as previously described unless all their arguments are of type rational fraction.

If all arguments are type rational fractions, the result is given type rational fraction and formed as described below. What follows is strictly true only for expressions that do not contain nested parentheses. With more complicated expressions, the results are analagous, with some complicated exceptions that we will not describe here.

```
L = LSUM (L1, L2)

L = LDIFF (L1, L2)
```

L1 and L2 are type rational fractions.

We follow the model of Example 1. L1 and L2 are added (subtracted) over the least common multiple M of their denominators. In the numerator of L parentheses are removed. The denominator of L contains the known factors of M.

```
L = LPROD (L1, L2)
```

L1 and L2 are rational fractions. The numerator of L contains the factors of the numerators of L1 and L2. The denominator of L contains the factors of the denominators of L1 and L2. Like factors are collected.

L = LQUOT (L1, L2, SW)

L1 and L2 are rational fractions. L2 is inverted, and we form L1 · 1/L2 as in LPROD. SW = . T.

L = LDERIV (L1, 1HX)

L1 is a rational fraction. We follow the usual rule from elementary calculus that

$$\frac{D}{DX}\left(\frac{U}{V}\right) = \frac{V\,DU - U\,DV}{V^2} \quad .$$

The denominator of L contains all the factors of the denominator of L1 with their exponents doubled. In the numerator, all brackets are removed.

Contrast this with differentiation of SPASM series, where we use the rule

$$\frac{D}{DX}(UV^{-1}) = -UV^{-2}\,\frac{DV}{DX} + V^{-1}\,\frac{DU}{DX} \quad .$$

For reduction to lowest terms, we provide

L = LRLT (L1)

If L1 is a rational fraction, we form the greatest common divisor of each factor of the numerator and each factor of the denominator. The fraction is reduced to lowest terms by division of numerator and denominator by each greatest common divisor found.

If L1 is not a rational fraction, each term of L1 is treated as a rational fraction and the resulting terms summed. LRLT is slow.

4.3 Trigonometric Series

Finite trigonometric series of the form

$$\sum_i (A_i) \sin (B_i) + \sum_j (C_j) \cos (D_j)$$

have many desirable properties. They are closed under addition, subtraction, multiplication, and differentiation; and with suitable restriction on the A's, B's, C's, and D's, they are closed under integration. Moreover, any polynomial $P(\sin \alpha_1, \ldots, \sin \alpha_n; \cos \beta_1, \ldots, \cos \beta_m)$ can be uniquely represented in the above form. For all these reasons, such series have been prominent in classical analysis. For manipulating such series, we have made the special provision described below.

4.3.1 Definitions

We say that $\sum_i (A_i) \sin (B_i) + \sum_j (C_j) \cos (D_j)$ is a SPASM Trigonometric Series (STS) if

1. All A's, B's, C's, D's are SPASM expressions.
2. All B's are distinct and no B is zero.
3. All D's are distinct.
4. No A nor C is zero.
5. The leading coefficient of all B's and D's is nonnegative.

Example 5

$$(6) \sin (A + B) + (9) \cos (A + B)$$

and

$$(3X^2 + 4Y) \sin (A^2 + B^2) + (6Y) \cos (0)$$

are STS's, but

$$\sin^2 (X) + \cos^2 (X) \quad ,$$

$$\sin (X) \cos (Y) + \cos (X) \sin (Y) \quad ,$$

and

$$\sin(X) + \sin(X)$$

are not.

We further state that an STS has property I_x (integrable with respect to X) if

6. X does not appear in A or C in parentheses.

7. X does not appear in B or D in parentheses and B and D are either linear in X or independent of X.

Example 6

$$(6X^2)\sin(X + Y^2)$$

and

$$(6X^3(Y + Z) + \sin(Z))\sin(X + Y^2)$$

have property I_x but not I_y.

4.3.2 Operations

```
LT = LMAKTS (L)
```

L is any SPASM expression. LT is the equivalent STS. For example, if

$$L = (A)\sin^2(X) + (2A)\sin(X)\cos(X) + (A)\cos^2(X) \quad ,$$

then

$$LT = (A)\cos(0) + (A)\sin(2X) \quad .$$

```
LT = LTSUM (L1, L2)

LT = LTDIFF (L1, L2)
```

L1 and L2 are STS's. LT is an STS that is the sum (difference) of L1 and L2. For example, if

L1 = (1) cos (0) + (Z) sin (2X)

and

L2 = (Y) cos (0) + (Z + Y) sin (2X) ,

then

LT = (1 + Y) cos (0) + (2Z + Y) sin (2X) .

LT = LTPROD (L1, L2)

L1 and L2 are STS's. LT is the STS that is the product of L1 and L2. For example, if

L1 = (Y) cos (0) + (Y) sin (2X)

and

L2 = (Y) cos (0) - (Y) sin (2X) ,

then

LT = $(Y^2/2)$ cos (0) + $(Y^2/2)$ cos (4X) .

LT = LTDERIV (L, VAR)

L is an STS, and VAR is in atom format. We form the STS LT = $\partial L/\partial$(VAR). For example, if L = (X - A) sin (X), A depends on X, and VAR = ATOM (1HX), then

LT = (1 - DERIV (A, X, 1)) sin (X) + (X - A) cos (X) .

LT = LTINTEG (L, VAR)

L1 is an STS and VAR is in atom format. We form the STS LT = $\int$ L d(VAR). The result is correct if L has property I_{VAR}.

For all the above operations, if L1, L2, L have property I_x, then LT has property I_X.

L = LTSTE (S)

Given the STS S, convert it to SPASM expression L.

L = LTXPAND (S)

Given STS S, LEXPAND all coefficients and arguments to form L.

L = LTSUBST (S, VAR, L1)

Given STS S, LSUBST VAR for L1 in each coefficient and argument of S to form L.

L = LJGL (L1, LCOF, LARG, C2, C3, A2, A3)

is a general-purpose routine, which subsumes LTXPAND and LTSUBST. LJGL forms the STS L by applying the user-supplied functions LCOF and LARG to each term of L1. More precisely corresponding to each term of the form (C) SCN (A) (where SCN (A) means sin (A) or cos (A)) of L1, L contains a term of the form

(LCOF (C, C2, C3)) SCN (LARG (A, A2, A3)) ,

where C2, C3, A2, A3 are parameters. For example,

L = LJGL (L1, LSUBST, LSUBST, VAR, S, VAR, S)

does exactly the same thing as L = LTSUBST (L1, VAR, S).

There is an alternate entry

L = LJGLS (L1, LCOF, LARG, C2, C3, A2, A3)

which does exactly the same thing as LJGL except that the terms of L1 are erased as L1 is traversed. This is useful when storage is at a premium.

5. MISCELLANEOUS

5.1 Erasure

Let us reemphasize the importance of prompt erasure of expressions. Consider the two loops below:

```
      READ 2, L                      READ 2, L
2     FORMAT (I3)              2     FORMAT (I3)
      I = 0                          I = LDECODE (X, X, 2H0$)
      J = 1                          J = LDECODE (X, X, 2H1$)
      D0 1 K = 1, L                  D0 1 K = 1, L
1     J = I + J                1     J = LSUM (I, J)
```

There is a subtle distinction between these two programs. The storage required by the first program is independent of the value of L. This is <u>not</u> true of the second program!

In the statement J = I + J, the contents of I are added to the contents of J and the result placed in J. In the statement J = LSUM (I, J), the expressions pointed to by the contents of I and J are added and a pointer to their sum replaces the contents of J. The expression originally pointed to by J has not been erased; rather, a pointer to it has been overwritten.

To erase the intermediate results, it suffices to rewrite the right-hand loop as

```
      D0 1 K = 1, L
      JJ = J
      J = LSUM (I, J)
1     CALL ERASE (JJ)
```

Other ways of erasing are described below.

5.2 Nesting SPASM Functions

Those SPASM functions that return a list as their value may be called by an alternate name in which the initial L of the function name is replaced by an N, mnemonic for "nest." Such a call will do what the primary call does, with the addition that the name of the result list is pushed down on the system list NLIST. The statement CALL SAVEN causes a "barrier" to be pushed down on NLIST. The statement CALL EMPTYN causes all lists on NLIST down to the first barrier to be erased. The statement CALL RESTORN causes all lists down to the first barrier to be erased, and the barrier removed. A subroutine that uses this feature ought at least to begin with a call to SAVEN and end with a call to RESTORN, but this does not exhaust their uses.

This feature may be used to alleviate coding calls to ERASE. A more important role is that it allows function calls to be nested, and their result lists eventually to be erased. For example,

```
L = LSUM (NSUM (L1, L2), L3)
CALL EMPTYN
```

will leave the sum of L1, L2, and L3 in L, and the sum of L1 and L2 will be erased by the call to EMPTYN.

The user can also

```
CALL N (L)
```

which puts L on NLIST.

5.3 Equivalent and Identical Expressions

By identical (≡) SPASM expressions, we mean those expressions that if printed with LPRINT would look exactly the same. With this definition of identity, $(X + Y)(X - Y) \not\equiv X^2 - Y^2$.

We stress this point because equivalent mathematical expressions have many representations that are not identical. The user is often concerned with determining when two expressions are equivalent. The conditional transfer (ZTEST) in SPASM can only detect identity. The reason for this gap is the well-known discouraging result: given the possible SPASM expressions, there exists no finite algorithm for determining equivalence. All is not lost, however, because many particular types of expressions can be tested for equivalence. The main tool is the function ZTEST:

```
V = ZTEST (L)
```

L is a SPASM expression. V = .T. if L is identical to zero; otherwise, V = .F.

For the following special types of expressions, equivalence can always be determined:

1. Polynomials. V = ZTEST (NEXPAND (NDIFF (L1, L2))). If the polynomials do not contain parentheses, the call to NEXPAND can be deleted.

2. Rational fractions. V = ZTEST (NEXPAND (NCLEAR (NDIFF (L1, L2)))).

3. SPASM series. Convert to rational fractions by NRATIO and test as in 2.

4. SPASM trigonometric series with polynomial arguments and coefficents. V = ZTEST (NTXPAND (NTDIFF (L1, L2))).

5.4 Analyzing Expressions

```
Z = EXPH (L1, VAR)
Z = EXPL (L1, VAR)
```

Z is the highest (lowest) power of unparenthesized instances of VAR in L1. VAR must be in atom format.

L = LCOEF (L1, VAR, EXP)

L is that expression that is the coefficient of the unparenthesized instances of VAR ** EXP in L1. VAR must be in atom format. For example, if

$$L1 = X^2Y + X^2Y^2 + (1 + X^5)^2\ Y^{-3} \quad ,$$

and if

L2 = LCOEF (L1, ATOM (1HX), EXPH (ATOM (1HX), L1)) ,

L3 = LCOEF (L1, ATOM (1HY), EXPL (ATOM (1HY), L1)) ,

then

$$L2 = Y + Y^2$$

$$L3 = (1 + X^5)^2 \quad .$$

5.5 Converting FORTRAN Variables to SPASM Expressions

As we have seen, it makes no sense to have arithmetic statements containing both SPASM expressions and FORTRAN variables. We already know how to convert SPASM expressions to FORTRAN constants by using EVAL. Sometimes we want to make SPASM expressions of FORTRAN constants.

L = LCONS (N, D)

Recall from Section 3.1 that coefficients (and thus expression constants) are either rational fractions or floating-point constants. LCONS forms these two types of expression constants according to the following conventions (N and D can be any combination of TYPE REAL or INTEGER):

D = 0		LCONS	is floated (if necessary) N ,
D ≠ 0	N and D integers	LCONS	is rational fraction N/D reduced to lowest terms ,

otherwise, N and D are floated (if necessary), LCONS is set to their quotient.

For example, suppose we are given the SPASM expression L and wish to form the n SPASM expressions M(I) = (I/I + 1) L, I = 1,2,. . . , N. We could write

```
      CALL SAVEN
      D0  99  I = 1, N
99    M(I) = LPROD (NCONS (I, I + 1), L)
      CALL RESTORN
```

6. PRACTICAL CONSIDERATIONS

6.1 Initialization

Every SPASM program must begin by calling INIT, which initializes the list-processing system SLIP and sets up various SPASM bookkeeping arrangements. Then, practically every program must call RDEF, which reads control cards defining function names, the derivatives of functions, and dependency relationships. Also, various limits are imposed by considerations of storage, which may be reset by the user via labeled common blocks. These labeled common blocks are defined by SPASM in subroutine INIT and loaded via DATA statements. To reset them, the user should define labeled common blocks of the same name in his main program, ensure that it is loaded before INIT, and reload the variables via arithmetic replacement statements.

6.2 Definitions

Subroutine RDEF reads control cards defining function names, function derivatives, and dependency relationships. These control cards are identified by columns 1 to 10 and contain two other fields. The first field begins with the first nonblank character after column 10 and ends with the next blank. The second field begins with the first nonblank character after the first field and ends with the next blank. The fields are function names, integers, or atomic variables, as follows:

ID	First Field	Second Field
FUNCTION	symbol	integer
PARTIAL	symbol	integer
DEPEND	variable	variable
ENDEF	none	none

A FUNCTION card defines the first field as a function name, where the second field is the number of its arguments. A PARTIAL card signals that the model for the derivative of the function named in the first field with respect to its argument specified in the next field is next in the card deck. A DEPEND card defines the first field as dependent on the second field. An ENDEF card signals the end of definitions. The following deck defines SIN, COS, and their derivatives and makes y depend on x:

```
FUNCTION        SIN     1
FUNCTION        COS     1
PARTIAL         SIN     1
COS (ARGA) $

PARTIAL         COS     1
-SIN (ARGA) $
DEPEND          Y       X
ENDEF
```

6.3 Truncation of Series

The user can cause expressions to be truncated by initializing the labeled common block TRUNC.

```
COMMON/TRUNC/ITRUNC, TRUNC (2, N)
```

TRUNC (1, I) contains a word of the form ATOM (1HX), and TRUNC (2, I) contains a floating-point number, where I = 1, N. ITRUNC contains N in integer form. Then, in the expression resulting from an arithmetic operation or an input operation, any term that contains one of the variables in TRUNC to a power higher than the corresponding number in TRUNC will be dropped.

If the result of an operation would be

$$L = 1/4 * z^2 - 1/9 * y^2 + 1/8 * x * z - 1/2 * x * y$$

but TRUNC (1,1) = ATOM (1HZ) and TRUNC (2,1) = 1., then the result will be

$$L = -1/9.* y^2 + 1/8 * x * z - 1/2 * x * y \quad .$$

6.4 Field-Length Control

COMMON/LFLD/LFLD, LSTOR, LIMFLD, INCFLD

Blank common is used by SLIP for its list storage, because it is capable of being expanded upward. The length of blank common is initialized at LSTOR words, and the job's field length is adjusted to fit. The length of the job's field is maintained in LFLD. If SLIP runs out of storage, the job's field length is increased by INCFLD words. However, if this new value exceeds LIMFLD, the job aborts. The nominal values are

$$\text{LSTOR} = 4000_8$$

$$\text{LIMFLD} = 100000_8$$

$$\text{INCEED} = 4000_8$$

If the user wishes to change these parameters, he should do so via arithmetic replacement statements before calling INIT.

6.5 Work-Space Control

COMMON/WORK/LISTWK, INWK, NEXTWK, LIMWRK, WORK (1000)

The array WORK is used by many SPASM routines for temporary storage. LIMWRK contains the length of WORK, nominally 1000 decimal. The user may cause WORK to be loaded with a different size by defining this common block with a different dimension in his main program. However, he must also reload LIMWRK with the new size.

6.6 Exponent-Range Control

```
COMMON/BLKDEF/NXPW, NBPX, IRMND, MSKEXP
```

The arithmetic operations gain economies of space and time by packing the exponents of several variables in one word. NXPW contains the number of exponents per word, NBPX contains the number of bits per exponent, IRMND contains the number of unused bits per word, and MSKEXP is a mask containing NBPX bits right-justified. SPASM will abort if the absolute value of any exponent exceeds 2 ** (NBPX - 1) - 1. If the user changes any of these numbers, he must change them all consistently. The nominal values are

```
NXPW   = 7
NBPX   = 8
IRMND  = 4
MSKEXP = 377B
```

Thus, the nominal range of exponents is $-127 \leq \text{exp} \leq 127$.

6.7 Debugging

The principal problem in debugging a SPASM program is the timely erasure of expressions. If expressions are not erased soon enough, storage may be exhausted. If expressions are erased too many times, the job will continue for a while, and then abort mysteriously at a random later time.

```
CALL LISTS (DUMP)
```

causes a chart to be printed of all unerased lists and of the addresses of names of unerased lists. If DUMP is TRUE, the contents of lists are dumped in octal format. The user may first CALL MOPUP, which erases all system lists, leaving only the user's lists, but then the job cannot continue.

If the user erases an expression too many times, the job does not abort immediately, because of the way SLIP processes erasures. When the job finally does abort, there is no evidence as to where the error occurred. A switch is provided that if set TRUE will cause the job to abort when the error is committed. This feature should be used with discretion, because it severely increases machine time. The switch is set by defining the labeled common block

```
COMMON /DEBUG/DEBUG
```

and executing the statement

```
DEBUG = . T.
```

6.8 Loading

SPASM and SLIP together comprise upward of 150 routines, and the calling hierarchy goes down many levels. Also, it is normal that only a subset of the available routines should be needed by a given job. Program SELOAD was written by the author of SPASM because, although it is of general interest and independent, it is practically indispensable for this application. SELOAD is on the SAO system tape and is called by a program control card with three arguments.

```
SELOAD (LGO, SPASM, SELECT)
```

LGO is the file containing the user's program, SPASM is the file containing all the SPASM and SLIP routines, and SELECT is a file on which SELOAD will put the routines from SPASM required by the user's program. The following cards are the complete minimum set of SCOPE control cards for a SPASM job:

```
NHALL, G68095, CM60000, T100.
RUN (S)
REQUEST SPASM, HI.
SAO 2143 RING OUT
```

(Consult the Programmer of the Day for the current SPASM tape number.)

RFL, 60000.
SELOAD (LGO, SPASM, SELECT)
UNLOAD (SPASM)
CLEAR.
LOAD (LGO)
LOAD (SELECT)
EXECUTE.
(end of record)

6.9 Sample SPASM Job

```
NORTON,G68095,CM60000,T100.
RUN(S)
REQUEST SPASM,HI.  SAO 2143  RING OUT
RFL,60000.
SELOAD(LGO,SPASM,SELECT)
UNLOAD(SPASM)
CLEAR.
LOAD(SELECT)
LOAD(LGO)
EXECUTE.

(END-OF-RECORD)

      PROGRAM MAIN(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
       DIMENSION L(20)
      PRINT 2
2      FORMAT(*)SAMPLE SPASM OUTPUT.*//)
      CALL INIT
      CALL RDEF
      L(1) = LREAD(LABEL,OK)
      CALL LPRINT(L(1),5,10,LABEL)
      IF(OK)GO TO 10
      CALL ARNORML
10    L(2) = LDERIV(L(1),ATOM(1HX))
      CALL LPRINT(L(2),5,10,10HDSAMPLE/DX      )
      L(3) = LDECODE(LABEL,OK,10H 1=A/B,$      )
      L(4) = LSUBST(L(2),ATOM(1HY),L(3))
      CALL LPRINT(L(4),5,10,10HANSWER          )
      DO 20 I = 1,4
20    CALL ERASE(L(I))
      CALL MOPUP
      CALL LISTS(.T.)
      END

(END-OF-RECORD)

FUNCTION   SIN 1
FUNCTION  COS 1
PARTIAL     SIN 1
 COS(ARGA)$
PARTIAL      COS 1
-SIN(ARGA)$
DEPEND       Y   X
ENDEF
  SAMPLE = 1/2*X**2*Y + SIN(X**2) $

(END-OF-FILE)
```

6.10 Sample SPASM Output

```
FUNCTION    SIN 1
FUNCTION   COS 1
PARTIAL      SIN 1


COS(ARGA)
PARTIAL        COS 1


-1*SIN(ARGA)
DEPEND         Y    X
ENDEF

                          2          2
SAMPLE       = SIN(X )+1/2*X *Y

                                    2                2
DSAMPLE/DX = 2*X*COS(X )+X*Y+1/2*X *DERIV(Y,X,1)

                                 2
ANSWER       = X+2*X*COS(X )-1*A/B*X

THERE ARE      0 SLIP CELLS IN USE AND  1024 AVAILABLE.

 LOC(HDR)  LOC(DL) REF CNTR

LOC(NAME) LOC(LIST)

END LIST NAMES.

READER         LIST
```

APPENDIX

EXPRESSION SYNTAX

This appendix defines the syntax of SPASM expressions as they are punched into data cards. The elements of the language are as follows:

letters

decimal digits

decimal point

arithmetic operators — +, -, *, /, **

matched pairs of brackets

delimiters — =, blank, comma, $

symbols — a string of letters and digits, beginning with a letter, preceded by anything else and followed by anything other than a letter or a digit. If there are more than five characters, only the first five are used.

numbers — a string of decimal digits and at most one decimal point, preceded by anything other than a digit or a letter and followed by anything other than a digit or a decimal point.

integers — a number that does not contain a decimal point.

subexpressions — an expression, as defined below, enclosed in a matched pair of brackets.

argument list — one or more expressions enclosed in brackets and separated by commas.

functions — a symbol followed by an argument list. However, the symbol must have been previously defined to be a function name.

atomic variables — symbols not followed by argument lists.

exponents	- ** followed by an integer.
factors	- an atomic variable, a subexpression, or a function, which may be followed by an exponent.
terms	- a sequence of numbers and factors separated by * or /.
expression	- a sequence of terms separated by + or -, which may be preceded by + or -.
labels	- a string of not more than 10 characters other than = or $, followed by =.
record	- an expression followed by $, which may be preceded by a label.

The syntax is similar to that of a FORTRAN arithmetic replacement statement, with the following exceptions:

1. The left-hand side and = are optional.

2. Subscripts may not appear.

3. ** can only be followed by an integer.

4. Blanks serve to terminate fields. With this proviso, redundant blanks are allowed anywhere.

5. If an E is concatenated with a number in a coefficient (as opposed to an exponent), it is interpreted as part of the number, in FORTRAN E format. To avoid this interpretation, a * or a blank must separate the E from the number. This rule also applies when the initial letter of a multicharacter name is E.

INDEX
(by SPASM operators)

Manipulating Expressions

Changing Expression Types

Analyzing Expressions

Storage Management

Formatting

BIOGRAPHICAL NOTES

NORTON M. HALL received the A. B. degree from Stanford University in 1956.

Before joining the programing staff at Smithsonian Astrophysical Observatory in 1966, Mr. Hall worked as a programer at the John Hancock Life Insurance Company and at the Massachusetts Institute of Technology. He is currently serving as Acting Manager of the Systems Division at SAO.

JEROME R. CHERNIACK received the B. S. degree from City College of New York in 1958.

Mr. Cherniack has been with the Smithsonian Astrophysical Observatory since 1959. He is currently working with the geophysics and geodetics group, concentrating on computer sciences and applied mathematics.

NOTICE

This series of Special Reports was instituted under the supervision of Dr. F. L. Whipple, Director of the Astrophysical Observatory of the Smithsonian Institution, shortly after the launching of the first artificial earth satellite on October 4, 1957. Contributions come from the Staff of the Observatory.

First issued to ensure the immediate dissemination of data for satellite tracking, the reports have continued to provide a rapid distribution of catalogs of satellite observations, orbital information, and preliminary results of data analyses prior to formal publication in the appropriate journals. The Reports are also used extensively for the rapid publication of preliminary or special results in other fields of astrophysics.

The Reports are regularly distributed to all institutions participating in the U. S. space research program and to individual scientists who request them from the Publications Division, Distribution Section, Smithsonian Astrophysical Observatory, Cambridge, Massachusetts 02138.